SANDIA REPORT
SAND98–1981
Unlimited Release
Printed September 1998

A Dense Out-of-Core Solver (DOCS) for Complex-Valued Linear Systems

Cullen E. Lee

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

SAND98-1981
Unlimited Release
Printed September 1998

A Dense Out-of-Core Solver (DOCS) for Complex-Valued Linear Systems

Cullen E. Lee
Special Projects Department II
Sandia National Laboratories
PO Box 5800
Albuquerque, NM 87185-0314

Abstract

Dense systems of linear equations are quite common in many science and engineering applications. Such linear systems place extreme storage and computational demands on computer resources and, in many cases, may severely limit the subsequent analysis. A dense out-of-core solver (DOCS) that operates on a partitioned coefficient matrix can reduce the in-core storage requirements of the linear system while spreading the associated computational burden over multiple processors (which reduces run time as well). In this report, I describe a DOCS that operates on a partitioned coefficient matrix that may be distributed over multiple external storage devices. I have implemented this solver using Message-Passing Interface (MPI) protocols. This report presents performance data from a series of run time trials that compare the run time of both sequential and parallel implementations of the DOCS.

Acknowledgment

The author would like to thank Janise Baldo-Pulaski for her help in organizing this report and Raymond Zazworsky for his technical discussions, which have undoubtedly improved the text.

Contents

Figures

Summary

Dense systems of linear equations are common in many science and engineering applications. Such linear systems place extreme storage and computational demands on computer resources and, in many cases, may severely limit the subsequent analysis. A dense out-of-core solver (DOCS) that operates on a partitioned coefficient matrix can reduce the in-core storage requirements of the linear system while spreading the associated computational burden over multiple processors (which reduces run time as well).

In this report, I describe a DOCS that operates on a partitioned coefficient matrix that may be distributed over multiple external storage devices. I have implemented this solver using Message-Passing Interface (MPI) protocols. This report also presents performance data from a series of run time trials that compare the run time of both sequential and parallel implementations of the DOCS.

Using the parallel version of the DOCS algorithm, the run time trials demonstrated nearly linear speedup. This version of the algorithm, as currently implemented using MPI protocols, is suited for a strictly parallel computing environment, but can be easily modified for use in a distributed computing environment as well.

The author can be reached at celee@sandia.gov.

Nomenclature

BSS	backward solve sequence
DOCS	dense out-of-core solver
EM	electromagnetic
FSS	forward solve sequence
I/O	input/output
MPI	Message-Passing Interface
RAM	random access memory
RHS	right-hand side

A Dense Out-of-Core Solver (DOCS) for Complex-Valued Linear Systems

Introduction

Dense linear systems result from a wide range of applications and especially those applications in which all-pair interactions occur. Such linear systems are commonplace in problems involving electromagnetic (EM) scattering in the presence of electrically large objects. In fact, the popularity of asymptotic methods such as ray-based methods in EM analysis probably stems from the demands that dense linear systems place on the analyst. Unfortunately, in electromagnetics, there is a class of scattering problems for which asymptotic methods simply fail and one is forced to consider more exact methodologies, such as those based upon Method-of-Moments [1, 2]. In EM applications, moment method procedures generally yield dense and complex-valued linear systems whose dimension is proportional to the electrical size of the source current support region. Such linear systems place demands on the analyst that can quickly exceed available computing resources. Subsequently, this particular area of EM analysis has always been problematic for the analyst to deal with because it represents an area that requires both large amounts of storage and significant computing resources.

The primary purpose of this report is to present a dense out-of-core solver (DOCS) for large complex-valued linear systems of equations that would normally exceed the in-core capacity of the more commonly configured workstation. In this regard, the DOCS provides a cost effective means of extending the solution capability of a workstation that is already random access memory (RAM) constrained. Secondly, to effectively solve the very large linear systems, a parallel implementation of the DOCS algorithm is also now available (whose performance I will discuss later in this report).

Performance benchmarks for the sequential implementation of DOCS have already been provided in [3]. Consequently, this report will focus more on the parallel aspect of the DOCS. It will be shown that the most computationally intensive portion of the DOCS falls into a category favoring parallelization so that significant speedup is possible. However, simply applying multiple processors to this problem does not guarantee that the added processors will be used efficiently. Efficient use of this solver within a multiprocessor environment requires consideration be given to the partitioning of the coefficient matrix. Fortunately, within the DOCS, there is a clear path available for maintaining a *reasonable* level of load balance.

The following discussion presents the general problem and assumptions, describes the solution approach with a more detailed definition of the out-of-core algorithm, provides a discussion of the run time performance results, and examines the parallel performance of this algorithm. The appendices include implementation-specific details (see Appendix A for File Name Convention and Appendix B for Distributed Data Convention).

Problem Description

The DOCS described here solves the following matrix equation:

$$AX = B \qquad \textbf{Eqn 1}$$

where A is the $n \times n$ coefficient matrix, B is the $n \times k$ matrix of right-hand sides (RHS), and X is the $n \times k$ matrix of unknowns. Each of these matrices is assumed to be partitioned; that is, each represents a matrix of submatrices. In this case, A is a matrix of $m \times m$ submatrices, and X and B are each a matrix of $m \times 1$ submatrices. The coefficient matrix is assumed to be dense and can be complex valued.

Approach

Implementation of the DOCS is modeled after that of a standard LU decomposition of A; see [4,5] for an example. Consequently, the DOCS uses a two-step process that simultaneously factors and solves Eqn 1. This process can be conceptually understood by considering Eqn 1 in terms of two triangular systems, the first of which has the following form:

$$LY = B \qquad \textbf{Eqn 2}$$

where L is a lower triangular matrix.

The first step of the DOCS consists of a forward solve sequence (FSS) of m stages as indicated in Eqn 2. The i^{th} forward solve stage begins by factoring the diagonal submatrix, A_{ii}, into its L_{ii} and U_{ii} components, solves for the submatrices below (L_{ji}, $i < j \leq m$) and to the right (U_{ij}, $i < j \leq m$) of the diagonal submatrix, calculates the intermediate solution of $L_{ii}\, Y_i = B_i$ for the corresponding partition, and updates the unused partitions in the RHS and coefficient matrices before proceeding to the next submatrix on the diagonal. When all forward solve stages have been completed, the matrices U and Y are stored on the disk as components of the second triangular system given by

$$UX = Y \qquad \textbf{Eqn 3}$$

The last step of the DOCS consists of a backward solve sequence (BSS) of m stages as indicated in Eqn 3. The BSS is less complicated, but still involves a solve and update cycle similar to the FSS described previously. Beginning with U_{mm}, each matrix, U_{ii}, and its corresponding RHS, Y_i, is used to solve for X_i, which is then used to update the RHS (Y_j, $1 \leq j < i$) above the current row partition. At the end of this solution process, both the coefficient matrix and the RHS matrix will be overwritten.

Finally, it is also apparent that the overriding assumption in the implementation of the DOCS is that at the i^{th} forward solve stage, the diagonal submatrix A_{ii} of the coefficient matrix must possess a viable LU decomposition; otherwise, the process will fail.

The DOCS Algorithm

The DOCS algorithm is implemented using a sequence of fundamental operations that are performed on the submatrices of A. The two most computationally intensive of these operations are

- An LU decomposition $L_{ii}\ U_{ii} = A_{ii}$
- An outer product operation of the form $A_{ij} \leftarrow A_{ij} - L_{ik}\ U_{kj}$

Of these two operations, the outer product is performed much more frequently than is the LU decomposition, but can be optimized for a given host platform using compiler directives. The other matrix operations that compose the DOCS are the forward solve, the transpose, and the row permutation. Each of these matrix operations is apparent in the pseudo-code given here for the DOCS.

```
(FSS : Each Stage is Indexed by $i$)

For $i$ = 1 to m – 1

(Perform LU decomposition on $A_{ii}$ and calculate the $i^{th}$ intermediate solution.)

    • Retrieve $A_{ii}$ and $B_i$ from disk.
    • Obtain $L_{ii}\ U_{ii} = A_{ii}$ and $p_i$ from the LU decomposition, where $p_i$ is the row permutation
      vector.
    • Permute $B_i$ using $p_i$ and solve $L_{ii}Y_i = B_i$ for $Y_i$.
    • $B_i \leftarrow Y_i$ and $A_{ii} \leftarrow L_{ii}\ U_{ii}$ then write to disk.

(Solve $L_{ii}\ U_{ij} = A_{ij}$ for $U_{ij}$ where $j = i + 1, ..., m$)

    For $j = i + 1$ to $m$
        • Retrieve $A_{ij}$ from disk and permute using $p_i$.
        • Solve for $L_{ii}\ U_{ij} = A_{ij}$ for $U_{ij}$.
        • $A_{ij} \leftarrow U_{ij}$ and write to disk.
    End ($j$)

(Solve $L_{ji}\ U_{ii} = A_{ji}$ for $L_{ji}$ where $j = i + 1, ..., m$ then update $A$ and $B$)

    For $j = i + 1$ to $m$
        • Retrieve $A_{ji}$ and $B_j$ from disk.
        • Solve $L_{ji}\ U_{ii} = A_{ji}$ for $L_{ji}$.
        • $B_j \leftarrow B_j - L_{ji}Y_i$ and write to disk.
        • Update remainder of coefficient matrix.
        For $r = i + 1$ to $m$
            • Retrieve $A_{jr}$ and $U_{ir}$ from disk.
            • $A_{jr} \leftarrow A_{jr} - L_{ji}U_{ir}$ and write to disk.
        End ($r$)
    End ($j$)
End ($i$)
(FSS: $i = m$)
1. Retrieve $A_{mm}$ from disk.
2. Obtain $L_{mm}U_{mm} = A_{mm}$ and $p_m$ from the LU decomposition.
3. Permute $B_m$ using $p_m$ and solve $L_{mm}Y_m = B_m$ for $Y_m$.
4. Backsolve $U_{mm} X_m = Y_m$ to obtain $X_m$.
5. $Y_m \leftarrow X_m$ and write to disk.
6. Update $Y$.
    For $i = m - 1$ to 1
        • Retrieve $Y_i$ and $U_{im}$ from disk.
        • $Y_i \leftarrow Y_i - U_{im}X_m$ and write to disk.
    End ($i$)


(BSS: Each Stage is Indexed by $i$)
For $i = m - 1$ to 1
1. Retrieve $U_{ii}$ and $Y_i$ from disk.
2. Solve $U_{ii}X_i = Y_i$ for $X_i$.
3. $Y_i \leftarrow X_i$ on disk.
4. If $i > 1$ Then (Update $Y$)
        For $j = i - 1$ to 1
            • Retrieve $Y_j$ and $U_{ji}$ from disk.
            • $Y_j \leftarrow Y_j - U_{ji}X_i$ and write to disk.
        End ($j$)
    End If
End ($i$)
```

At the ith stage of the FSS, there is one LU decomposition, (m-i + 1) row permutations, (m-i) transposes, 2(m-i) + 1 forward solves, and $(m\text{-}i)^2 + (m\text{-}i)$ outer products. At the ith stage of the BSS, there is one backward solve operation and (m-i) outer product operations.

It is apparent that the computational load of the DOCS algorithm decreases at each stage of the FSS and at each stage of the BSS. Furthermore, the FSS is much more computationally intensive than is the BSS. This behavior is illustrated in Figure 1 for an *m* = 20 partition of a coefficient matrix of *n* = 4485 complex unknowns. The first 20 stages comprise the FSS and the last 20 stages comprise the BSS.

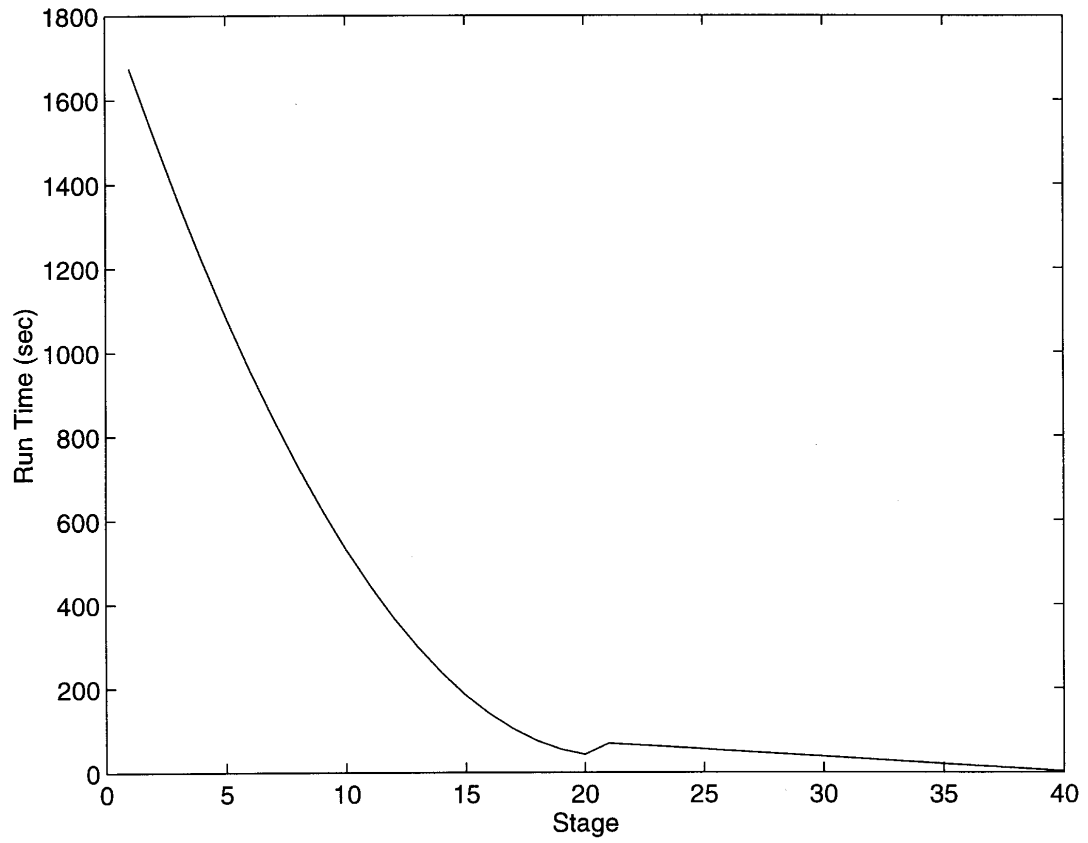

Figure 1 Single processor benchmark 1. *Single processor run time results for a coefficient matrix with n = 4485 complex unknowns, k = 180, and an m = 20 partition.*

The previous discussion illustrates the main trait of any out-of-core solver; that is, reduce the burden placed on the RAM at the expense of increased disk input/output (I/O) activity. On single-processor platforms, this corresponds to an increase in solution time [3]. The amount of overhead associated with disk I/O is proportional to the partitioning of the coefficient matrix; in other words, as *m* increases, so does the disk I/O overhead. Even so, on a single-processor platform, it is possible to minimize the solution time (within the context of the DOCS) by minimizing the partition dimension for the available RAM on the host platform. This is illustrated in Figure 2, where I used two different partitions (*m* = 4 and *m* = 20) for solving a series of linear systems.

Fortunately, the structure of the DOCS algorithm lends itself to the integration of parallel constructs. By performing this integration, it is possible to more than offset the inherent increase in disk I/O activity if there is sufficient communication bandwidth to the external disk drives.

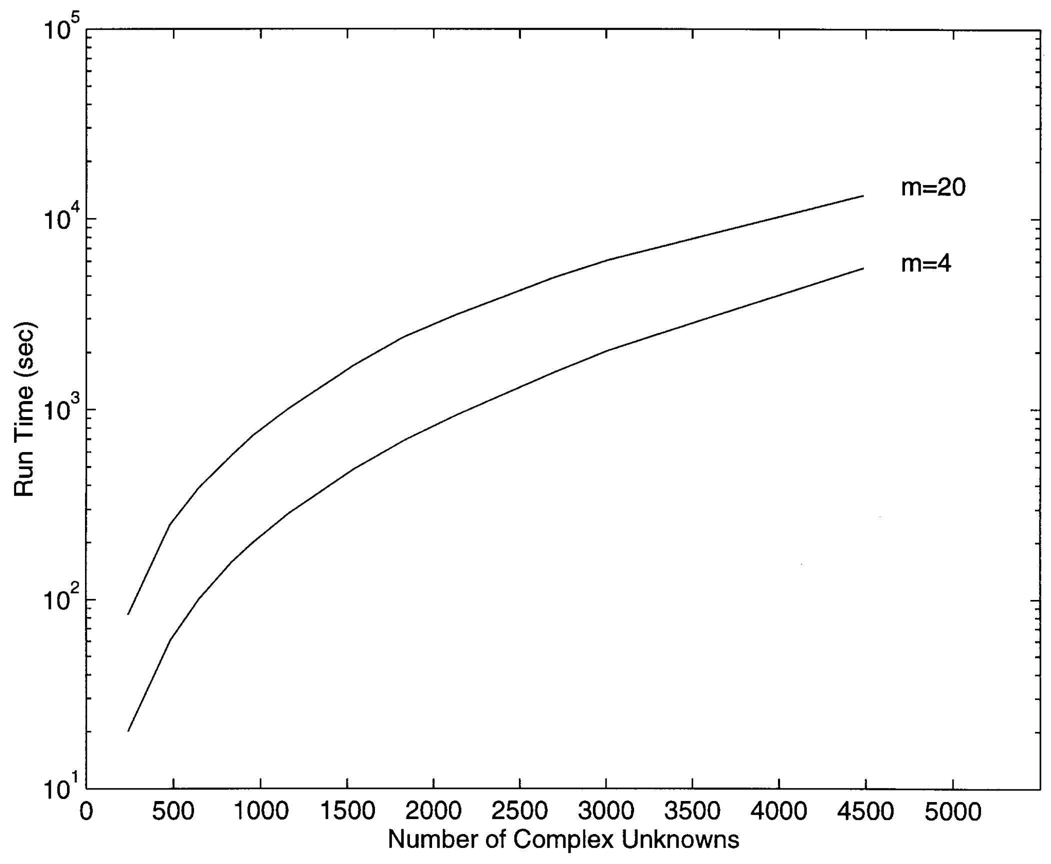

Figure 2 Single processor benchmark 2. *Single processor run time results for k = 180 with m = 4 and m = 20 partitioning of the coefficient matrices.*

Parallel Performance

I parallelized the DOCS algorithm using the Message-Passing Interface (MPI) [6]. In fact, I have extensively tested the algorithm using two different implementations of MPI. I originally tested with an MPI version called *mpich*, available from the World Wide Web. However, I generated all the data presented here using the Silicon Graphics® implementation.

The single most computationally intensive portion of the DOCS algorithm is associated with the computation of the outer products in the early stages of the FSS (see Figure 1). Fortunately, this set of operations can be performed in parallel at each stage of the FSS with a minimal amount of interprocessor communications. Consequently, most of the run time improvement achieved through parallelism will be realized at the early stages of the FSS. This result is shown in Figure 3, which is the multiprocessor analog of Figure 1.

The ideal result from parallelizing the DOCS would be to allow a significant reduction in solution time while simultaneously maintaining the load balance. This goal is difficult to achieve because there is a relatively small sequential component within the algorithm. That is, all processors must wait while the root processor computes a LU decomposition at the beginning of every forward solve stage and while it executes a backward solve computation at the beginning of every backward solve stage. Furthermore, an MPI barrier command is needed to synchronize all the processors before the coefficient matrix can be updated in the FSS.

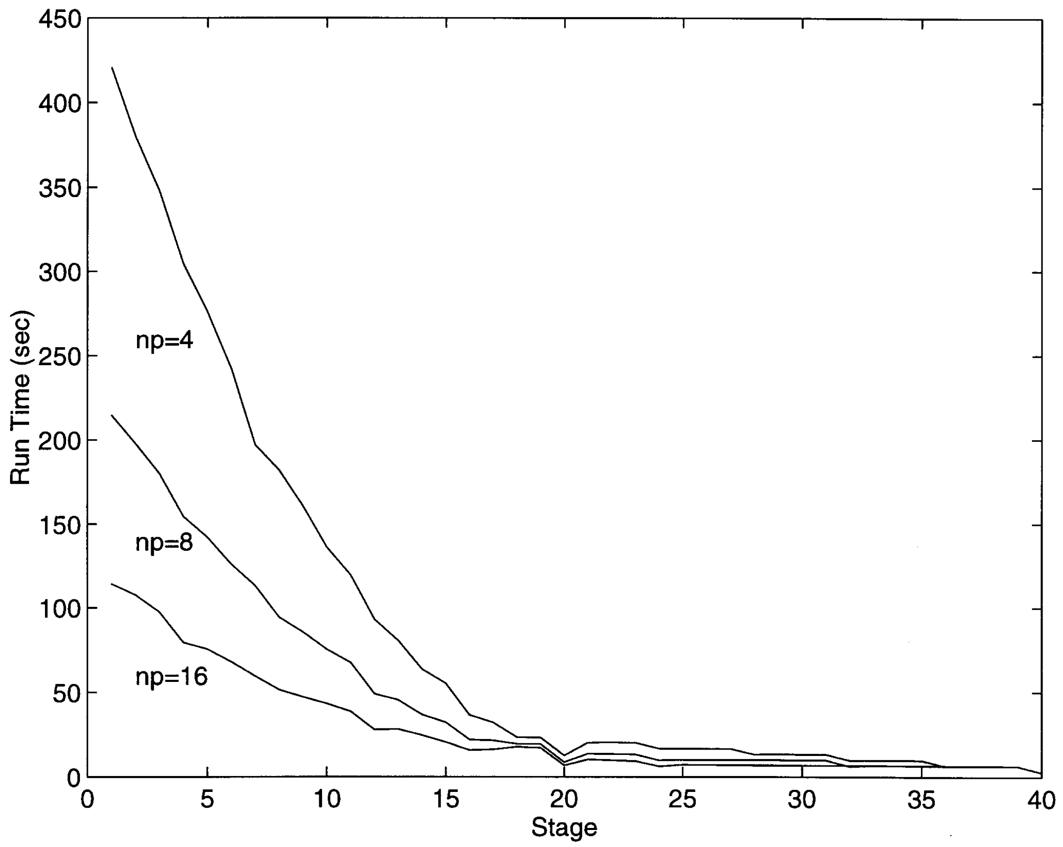

Figure 3 Multiple processor benchmark 1. *Run time results per stage for an m = 20 partitioning of a coefficient matrix with n = 4485 complex unknowns, k = 180, parameterized by the number of processors.*

The following are a set of guidelines for maintaining the load balance at a *reasonable* level for the DOCS:

- The number of processors, n_p, applied to the problem should satisfy $n_p \leq m$
- Assign the first $(m - 1) \times (m - 1)$ submatrices of the coefficient matrix the same dimension, s, such that the dimension, q $(q \leq s)$, of A_{mm} is as close to s as possible while maintaining $(m - 1)s + q = n$. The value of s that achieves this result is given by $s = \lceil n / m \rceil$ where $\lceil x \rceil$ denotes the ceiling function of x (the smallest integer larger than x).

One difference between the sequential and parallel implementations is now obvious. By reducing the partition dimension, one can effectively reduce the run time of the sequential implementation; however, in the parallel implementation, this is undesirable. This is apparent, considering that by increasing the partition dimension, the computational load (due primarily to the outer product operation) can be distributed over many processors without significantly increasing the interprocessor communication burden. On the other hand, when $n_p > m$ the problem cannot be effectively parallelized. As Figure 4 shows, there would be little reduction in run time.

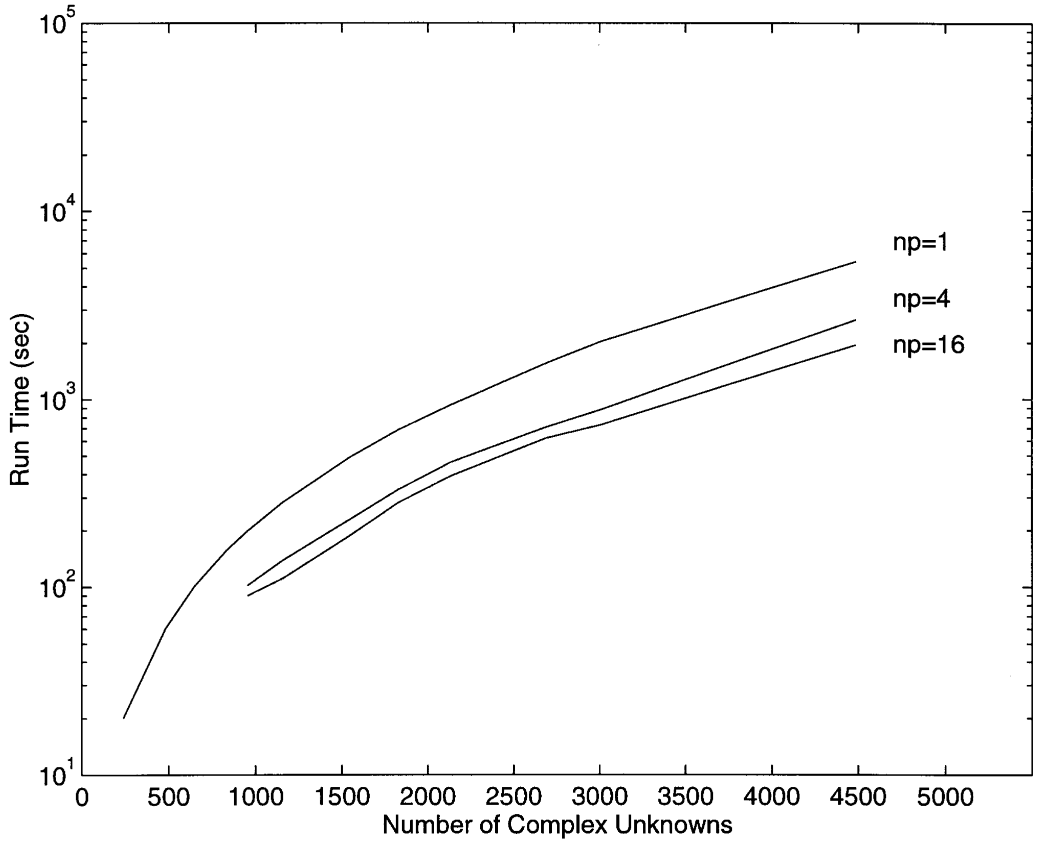

Figure 4 Multiple processor benchmark 2. *Run time results for an m = 4 partitioning of the coefficient matrices parameterized by the number of processors and k = 180.*

By increasing the partition dimension, the problem can be distributed over many more processors, and additional run time improvement is possible. In fact, as Figure 5 illustrates, *nearly linear speedup* is possible when operating in the regime $n_p \leq m$.

This approach to run time reduction cannot be continued without considering the effect it has on the solution process. In fact, there is a limit above which m must not be increased. This is evident by considering the fact that as m increases, the dimension of the submatrices of A decreases and the quality of the pivot elements used in the LU decomposition of the A_{ii} will be reduced, ultimately leading to the failure of the solution process. There are no rules governing how large m can become for a given coefficient matrix without resulting in a failure of the solution process, but it must certainly depend on the structure of this matrix. In that regard, applying the DOCS to a less-than-dense coefficient matrix should be done with a high degree of caution and certainly not without preconditioning the coefficient matrix.

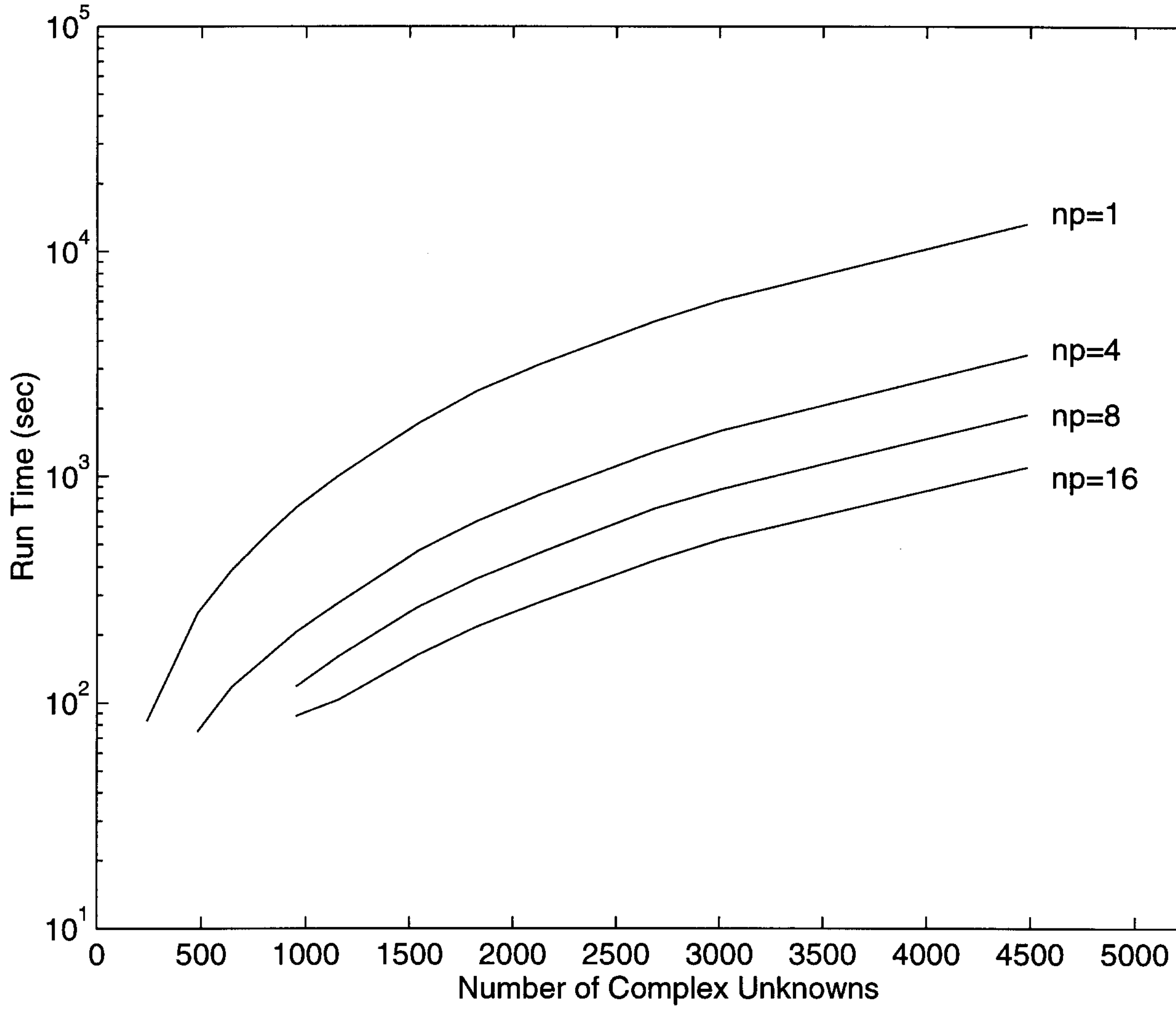

Figure 5 **Multiple processor benchmark 3.** *Run time results for an m = 20 partitioning of the coefficient matrices parameterized by the number of processors and k = 180.*

Finally, for analysis purposes, I have hosted the DOCS on a 4xR10000 Silicon Graphics® Origin200 system that was connected to a MegaDrive® 8x18GB storage device (RAID level 0). With this particular configuration, I can analyze complex linear systems up to 50,000 unknowns. Figure 6 illustrates the run time performance that I have realized with this configuration.

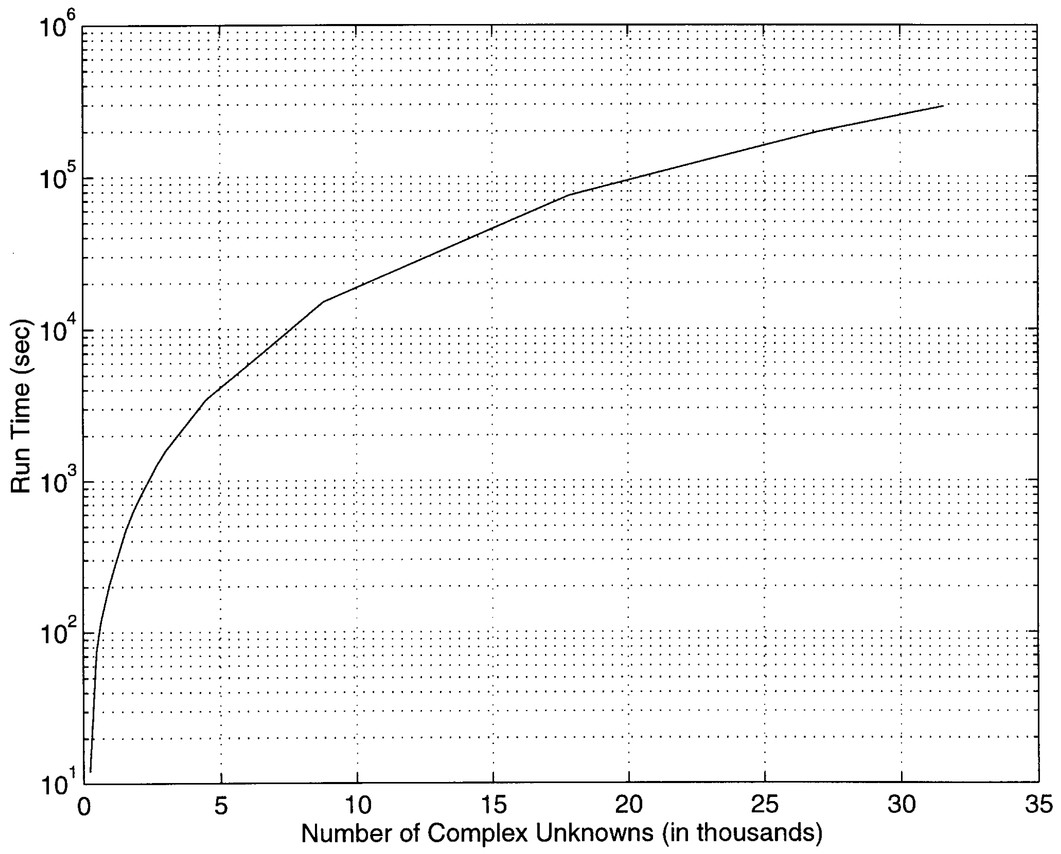

Figure 6 **Multiple processor benchmark 4.** *Run time results for an m=20 partitioning of the coefficient matrices with n_p=4.*

Summary and Conclusions

Although there are many science and engineering applications that require such a capability, solving large dense linear systems is a difficult and time-consuming task. It requires large amounts of external storage space and (preferably) a multiprocessor computing environment. The primary goal of the DOCS was to provide a cost-effective means of extending the solution capability of workstations that are already RAM constrained. A subsequent goal was to provide a means of effectively solving the very large dense linear systems ($n \geq 10^5$). This latter class of problems requires a multiprocessor computing environment connected to perhaps multiple sets of storage devices through a high-bandwidth communication channel.

I demonstrated nearly linear speedup using the parallel version of the DOCS algorithm. This version of the algorithm, as currently implemented using MPI protocols, is suited for a strictly parallel computing environment, but can be easily modified for use in a distributed computing environment as well.

I wrote the DOCS described here in Fortran 77; currently, both sequential and parallel versions of the code are available. Both versions of the code allow the user to distribute the coefficient matrix and RHS data over up to four unique external disk drive systems. Either code will be available, with all relevant routines and instructions on how to compile and run the code. The author can be reached at celee@sandia.gov.

References

1. R. Harrington, 1985, *Field Computation by Moment Methods* (Malabar, FL: Krieger).

2. J. J. Wang, 1991, *Generalized Moment Methods in Electromagnetics* (New York: John Wiley & Sons).

3. C. E. Lee and R. M. Zazworsky, March 1997, A Dense Out-of-Core Solver for Workstation Environments, *Proceedings of the 13th Annual Review of Progress in Applied Computational Electromagnetics*, Monterey, CA.

4. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, 1992, *Numerical Recipes in Fortran*, 2nd ed. (New York: Cambridge University Press).

5. Jennings and J. J. McKeown, 1992, *Matrix Computation*, 2nd ed. (New York: John Wiley & Sons).

6. W. Gropp, E. Lusk, and A. Skjellum, 1994, *Using MPI: Portable Parallel Programming with the Message-Passing Interface* (Cambridge: MIT Press).

APPENDIX A FILE NAME CONVENTION

The DOCS operates on a partitioned linear system whose data may be distributed over up to four unique external disk drives. Every submatrix of this linear system, including those of the RHS matrix, must be given a unique file name. If the prefix of the file name is fname, the convention used in this implementation of the DOCS is as follows.

Input File Names

Submatrices of the coefficient matrix will be named with the file name extension _a.*i*, where *i* designates a specific submatrix and is an integer such that $1 \le i \le m^2$. The assignment of these extensions is from left to right and top to bottom. In this manner, A_{11} is identified by fname_a.1, A_{12} by fname_a.2, A_{13} by fname_a.3,..., and A_{mm} by fname_a. m^2.

Submatrices of the RHS matrix are assumed to be named with the extension _b.*j* where *j* is an integer such that $1 \le j \le m$. The assignment of these extensions is from top to bottom. In this manner, B_1 is identified by fname_b.1, B_2 by fname_b.2,.., and B_m by fname _b.*m*.

Output File Names

Submatrices of the solution matrix will be named with the extension _x.*j*, where *j* is an integer such that $1 \le j \le m$. The assignment of these extensions is from top to bottom. In this manner, X_1 is identified by fname_x.1, X_2 by fname_x.2,..., and X_m by fname_x.*m*.

Intermediate File Names

There are a set of intermediate files (transparent to the user) associated with a permutation vector generated by the LU decomposition of the diagonal submatrices, A_{jj}, which can be identified by their file name extension _pvt.*j*, where *j* is an integer such that $1 \le j \le m$.

APPENDIX B DISTRIBUTED DATA CONVENTION

As mentioned previously, the submatrices of both the coefficient matrix and RHS matrix may be distributed over up to four unique external disk drives. However, there is a specific rule that the DOCS uses to determine the proper location of these submatrices. First, suppose the external disk drives are denoted disk0, disk1, disk2, and disk3.

The submatrices of the coefficient matrix are assumed to be distributed according to the rule

fname_a.*i* resides on disk*j*, where *j* satisfies the relation $j = (i) \bmod 4$.

Also, the submatrices of the RHS matrix are assumed to be distributed across this set of disk drives in a similar manner. That is, fname_b.*i* resides on disk*j*, where *j* satisfies the relation $j = (i) \bmod 4$.

Finally, the submatrices of the solution matrix will be distributed across the disk drives according to the rule

fname_x.*i* resides on disk*j*, where *j* satisfies the relation $j = (i) \bmod 4$.

DISTRIBUTION:

External

1 Steve Simonds
Silicon Graphics Inc.
6501 Americas Pkwy.
Suite 565
Albuquerque, NM 87110

1 Mimi Celis
Silicon Graphics Inc.
Mail Stop 580
2011 N. Shoreline Blvd.
Mountain View, CA 94043

1 Prof. Anthony Skjellum
Mississippi State University
Dept. of Computer Science
PO Box 9637
Mississippi State, MS 39762

1 Prof. Andrew J. Terzouli, Jr.
Air Force Inst. Of Tech.
PO Box 3402
Dayton, OH 45401

1 Charles Liang
6816 Dwight St.
Ft. Worth, TX 76116

1 Dwayne Car
The Boeing Company
P.O. Box 516, MC S064 2263
St. Louis, MO 63166

Internal

1	MS 0333	William C. Moffatt, 1803
1	MS 0482	Henry P. Fell, 2161
1	MS 0482	Jay B. Vinson, 2161
1	MS 0309	Janise Baldo-Pulaski, 2413
20	MS 0314	Cullen E. Lee, 2435
1	MS 0105	Robert B. Asher, 2435
1	MS 0417	Raymond M. Zazworsky, 5413
1	MS 1110	Richard C. Allen Jr., 9205
1	MS 1110	David E. Womble, 9222
1	MS 1109	Robert E. Benner Jr., 9224
1	MS 0865	William A. Johnson, 9542
1	MS 1152	Joseph D. Kotulski, 9542
1	MS 9018	Central Technical Files, 8940-2
2	MS 0899	Technical Library, 4916
2	MS 0619	Review and Approval Desk, 12690 for DOE/OSTI